In [1]:

```
from datetime import datetime, timedelta,date
import pandas as pd
%matplotlib inline
from sklearn.metrics import classification_report,confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from __future__ import division
from sklearn.cluster import KMeans
import matplotlib
import matplotlib.pyplot
```

In [2]:

```
import chart_studio.plotly as py

import plotly.offline as pyoff
import plotly.graph_objs as go
import plotly
import plotly.graph_objs as go
from plotly.offline import *
```

In [3]:

```
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split
```

In [4]:

```
import xgboost as xgb
```

In [5]:

```
pyoff.init_notebook_mode()
```

In [6]:

```
tx_data = pd.read_csv('transactions_company_103600030.csv')
```

In [7]:

```
tx_data = tx_data[tx_data['purchaseamount']>0]
```

In [8]:

```
tx_data.head(10)
```

Out[8]:

	id	chain	dept	category	company	brand	date	productsize	productmeasure	purchasequantity	purchaseamount
0	86246	205	26	2630	103600030	9787	2012-03-13	120.0	CT	2	4.98
1	86246	205	26	2628	103600030	9787	2012-03-25	1.0	RL	2	2.00
2	86246	205	26	2630	103600030	9787	2012-04-02	70.0	CT	3	4.17
3	86246	205	26	2634	103600030	4139	2012-04-05	6.0	RL	1	3.99
4	86246	205	26	2631	103600030	4139	2012-04-05	140.0	CT	1	7.89
5	86246	205	26	2615	103600030	9878	2012-04-06	18.0	CT	1	3.55
6	86246	205	26	2628	103600030	9787	2012-05-17	1.0	RL	1	1.00
7	86246	205	26	2634	103600030	4139	2012-06-11	4.0	RL	2	8.18
8	86246	205	26	2614	103600030	9878	2012-06-17	14.0	CT	1	2.99

	id	chain	dept	category	company	brand	date	productsize	productmeasure	purchasequantity	purchaseamount
9	86246	205	26	2615	103600030	9878	2012-06-17	18.0	CT	1	5.29

In [336]:

```
segmented_data = segmented_data[segmented_data['monetary_value5M']<segmented_data['monetary_value5M
'].quantile(0.99)]
```

In [337]:

```
kmeans = KMeans(n_clusters=7)
kmeans.fit(segmented_data[['monetary_value5M']])
segmented_data['CLVCluster'] = kmeans.predict(segmented_data[['monetary_value5M']])
```

In [342]:

```
segmented_data.tail()
```

Out[342]:

	id	recency	frequency	monetary_value	R	F	M	Cluster	monetary_value5M	segment_Almost Lost	segment_Best Customer	segment_Big Spenders
0	86246	-53	48	348.65	1	1	1	1	165.80	0	1	0
1	16412251	-65	35	137.64	1	1	1	1	110.92	0	1	0
2	16606739	54	10	111.00	3	1	1	2	111.00	1	0	0
3	17678929	-80	15	195.89	1	1	1	1	172.72	0	1	0
4	21024070	71	11	145.36	3	1	1	2	145.36	1	0	0

In [201]:

```
tx_merge = order_cluster('LTVCluster', 'm6_Revenue',tx_merge,True)
```

In [341]:

```
segmented_data = order_cluster('CLVCluster', 'monetary_value5M',segmented_data,True)
```

In [202]:

```
tx_merge.groupby('LTVCluster')['m6_Revenue'].describe()
```

Out[202]:

LTVCluster	count	mean	std	min	25%	50%	75%	max
0	80147.0	9.061868	5.308502	0.00	4.4800	8.490	13.330	19.73
1	47127.0	30.509769	7.126309	19.74	24.2000	29.880	36.350	44.50
2	29374.0	58.580341	9.135362	44.51	50.5500	57.650	66.180	76.41
3	19098.0	94.047808	11.332115	76.42	83.9800	93.020	103.500	116.06
4	12662.0	137.524262	13.590463	116.07	125.6800	136.320	149.010	163.39
5	8163.0	188.706904	16.075626	163.40	174.7550	187.420	201.985	220.07
6	5583.0	251.470962	20.039912	220.09	233.8150	249.820	268.190	290.11
7	3652.0	329.482755	24.993832	290.18	307.6325	327.055	350.145	377.61
8	2538.0	426.258176	31.634528	377.67	397.7750	422.850	451.515	489.18
9	1525.0	552.043213	41.124207	489.24	515.7700	546.510	586.580	633.40

In [203]:

```
tx_cluster = tx_merge.copy()
```

In [204]:

```
tx_cluster.head()
```

Out[204]:

	id	Recency	Frequency_x	Revenue_x	OverallScore	Segment	RecencyCluster	Frequency_y	Frequency	FrequencyCluste
0	2715453497	26	42	238.41	6	High-Value	5	42	42	
1	403263222	27	33	192.14	6	High-Value	5	33	33	
2	403418132	18	29	263.49	6	High-Value	5	29	29	
3	3522753788	2	45	248.80	6	High-Value	5	45	45	
4	403551237	14	21	217.99	6	High-Value	5	21	21	

In [322]:

```
segmented_data = pd.read_csv('All_RFM.csv')
```

In [323]:

```
segmented_data = segmented_data[segmented_data['frequency']>0]
```

In [324]:

```
segmented_data.shape
```

Out[324]:

```
(225172, 10)
```

In [325]:

```
segmented_data.head()
```

Out[325]:

	id	recency	frequency	monetary_value	R	F	M	segment	Cluster	monetary_value5M
0	86246	-53	48	348.65	1	1	1	Best Customer	1	165.80
1	86252	-25	28	188.88	2	1	1	Big Spenders	5	84.58
2	12262064	103	2	6.48	4	4	4	Lost Cheap Customers	4	6.48
3	12277270	128	2	5.98	4	4	4	Lost Cheap Customers	4	5.98
4	12332190	2	8	104.35	3	2	1	Big Spenders	5	104.35

In [326]:

```
segmented_data.groupby('segment')['monetary_value5M'].describe()
```

Out[326]:

segment	count	mean	std	min	25%	50%	75%	max
Almost Lost	5605.0	194.788817	416.232882	75.33	102.9800	141.29	211.83	28707.83
Best Customer	18929.0	727.788934	67558.420320	1.29	81.9200	126.40	218.28	9289036.97

segment	count	mean	std	min	25%	50%	75%	max
Big Spenders	30935.0	922.380809	115079.259896	0.99	82.2100	113.55	183.17	20183381.05
Lost Cheap Customers	26583.0	4.913449	2.694718	0.01	2.3900	4.59	6.99	9.99
Lost Customers	596.0	190.058171	211.918819	75.33	100.8825	134.83	207.08	3695.28
Loyal Customers	11183.0	44.736830	14.999497	1.99	33.5550	44.93	56.23	75.31
Others	118150.0	22.168743	16.401510	0.01	9.7800	17.91	31.48	75.31

In [309]:

```
tx_cluster.shape
```

Out[309]:

```
(209869, 15)
```

In [206]:

```
tx_cluster.head()
```

Out[206]:

	id	Recency	Frequency_x	Revenue_x	OverallScore	Segment	RecencyCluster	Frequency_y	Frequency	FrequencyCluste
0	2715453497	26	42	238.41	6	High-Value	5	42	42	
1	403263222	27	33	192.14	6	High-Value	5	33	33	
2	403418132	18	29	263.49	6	High-Value	5	29	29	
3	3522753788	2	45	248.80	6	High-Value	5	45	45	
4	403551237	14	21	217.99	6	High-Value	5	21	21	

In [207]:

```
tx_class = pd.get_dummies(tx_cluster)
```

In [327]:

```
segmented_data = pd.get_dummies(segmented_data)
```

In [328]:

```
segmented_data.shape
```

Out[328]:

```
(225172, 16)
```

In [329]:

```
segmented_data.head()
```

Out[329]:

	id	recency	frequency	monetary_value	R	F	M	Cluster	monetary_value5M	segment_Almost Lost	segment_Best Customer	segment_Big Spenders
0	86246	-53	48	348.65	1	1	1	1	165.80	0	1	0
1	86252	-25	28	188.88	2	1	1	5	84.58	0	0	1
2	12262064	103	2	6.48	4	4	4	4	6.48	0	0	0
3	12277270	128	2	5.98	4	4	4	4	5.98	0	0	0
4	12332190	2	8	104.35	3	2	1	5	104.35	0	0	1

In [335]:

```
tx_class.tail()
```

Out[335]:

	id	Recency	Frequency_x	Revenue_x	OverallScore	RecencyCluster	Frequency_y	Frequency	FrequencyCluster	Re
209864	3581656848	206	1	6.39	1	1	1	1	0	
209865	121519771	206	1	2.29	1	1	1	1	0	
209866	737442884	206	1	3.99	1	1	1	1	0	
209867	4630455927	206	1	2.99	1	1	1	1	0	
209868	3990899158	206	1	6.99	1	1	1	1	0	

In [119]:

```
tx_class.to_csv('tx_actual.csv')
```

In [83]:

```
tx_merge.shape
```

Out[83]:

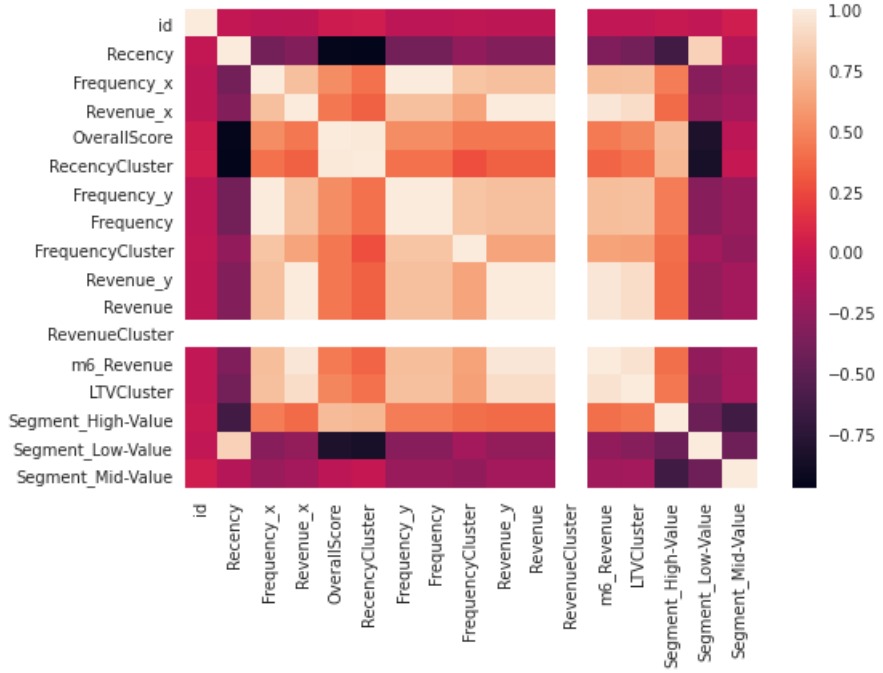
```
(209869, 11)
```

In [209]:

```
corr_matrix = tx_class.corr()
corr_matrix['LTVCluster'].sort_values(ascending=False)

sns.heatmap(tx_class.corr())
```

Out[209]:

```
<AxesSubplot:>
```



In [210]:

```
X = tx_class.drop(['LTVCluster', 'm6_Revenue'], axis=1)
```

```
X = tx_class.drop(['LTVCluster', 'm6_Revenue'],axis=1)
y = tx_class['LTVCluster']
```

In [343]:

```
X = segmented_data.drop(['CLVCluster', 'monetary_value5M'],axis=1)
y = segmented_data['CLVCluster']
```

In [357]:

```
X = segmented_data.drop(['CLVCluster', 'monetary_value5M'],axis=1)
y = segmented_data['monetary_value5M']
```

In [358]:

```
#X = tx_class.drop(['Revenue'],axis=1)
#y = tx_class['Revenue']

from xgboost import XGBClassifier
import xgboost as xgb
from sklearn import model_selection
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
```

In [359]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=56)
```

In [ ]:

```
ltv_xgb_model = xgb.XGBClassifier(max_depth=3, learning_rate=0.1,objective= 'multi:softprob',n_jobs
=-1).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'
       .format(ltv_xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.2f}'
       .format(ltv_xgb_model.score(X_test[X_train.columns], y_test)))
```

In [263]:

```
tx_class.groupby('Revenue').id.count()/tx_class.id.count()
```

Out[263]:

```
Revenue
0.00      0.000038
0.01      0.000014
0.03      0.000005
0.09      0.000010
0.10      0.000010
             ...
627.83    0.000005
631.09    0.000005
631.46    0.000005
632.18    0.000005
633.07    0.000005
Name: id, Length: 49339, dtype: float64
```

In [347]:

```
y_pred = ltv_xgb_model.predict(X_test)
```

In [348]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97     35220
           1       0.89      0.88      0.89     15895
           2       0.84      0.86      0.85      8665
           3       0.81      0.86      0.83      4729
           4       0.78      0.84      0.81      2590
           5       0.78      0.83      0.80      1433
           6       0.84      0.88      0.86       723

    accuracy                           0.92     69255
   macro avg       0.85      0.87      0.86     69255
weighted avg       0.92      0.92      0.92     69255
```

In [92]:

```
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn import model_selection
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC

xgb = xgb.XGBClassifier()
logreg2= LogisticRegressionCV()
knn = KNeighborsClassifier()
svcl = SVC()
adb = AdaBoostClassifier()
dtclf = DecisionTreeClassifier()
rfclf = RandomForestClassifier()

# prepare configuration for cross validation test harness
seed = 42
# prepare models
models = []
models.append(('LR', LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=5, multi_class='auto'))
)
models.append(('XGB', XGBClassifier()))
models.append(('KNN', KNeighborsClassifier(5)))
models.append(('DT', DecisionTreeClassifier(max_depth=5)))
models.append(('RF', RandomForestClassifier(n_estimators=100)))
models.append(('ADA', AdaBoostClassifier()))
models.append(('SVC', SVC(gamma='scale')))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=seed)
    cv_results = model_selection.cross_val_score(model,X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.779869 (0.000650)
XGB: 0.955772 (0.000938)
KNN: 0.752973 (0.001468)
DT: 0.955317 (0.000856)
RF: 0.954513 (0.001136)
ADA: 0.921728 (0.001114)
SVC: 0.779869 (0.000650)
```

In [93]:

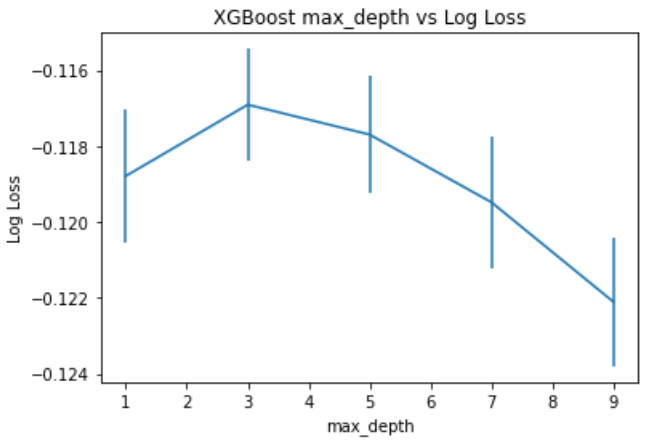
```
# XGBoost on Otto dataset, Tune max_depth
from pandas import read_csv
from xgboost import XGBClassifier
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot
# load data
data = pd.read_csv('transactions_company_103600030.csv')
dataset = data.values
# split data into X and y
X = tx_class.drop(['LTVCluster','m6_Revenue'],axis=1)
y = tx_class['LTVCluster']
# encode string class values as integers
label_encoded_y = LabelEncoder().fit_transform(y)
# grid search
model = XGBClassifier()
max_depth = range(1, 11, 2)
print(max_depth)
param_grid = dict(max_depth=max_depth)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold, verbose=
1)
grid_result = grid_search.fit(X, label_encoded_y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
 print("%f (%f) with: %r" % (mean, stdev, param))
# plot
pyplot.errorbar(max_depth, means, yerr=stds)
pyplot.title("XGBoost max_depth vs Log Loss")
pyplot.xlabel('max_depth')
pyplot.ylabel('Log Loss')
pyplot.savefig('max_depth.png')
```

```
range(1, 11, 2)
Fitting 10 folds for each of 5 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 24 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   18.3s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  2.3min finished
```

```
Best: -0.116904 using {'max_depth': 3}
-0.118784 (0.001739) with: {'max_depth': 1}
-0.116904 (0.001471) with: {'max_depth': 3}
-0.117691 (0.001541) with: {'max_depth': 5}
-0.119479 (0.001743) with: {'max_depth': 7}
-0.122103 (0.001701) with: {'max_depth': 9}
```



In [94]:

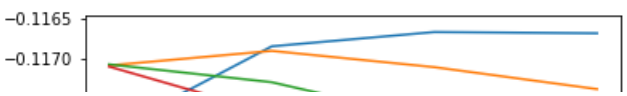
```
from pandas import read_csv
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
```

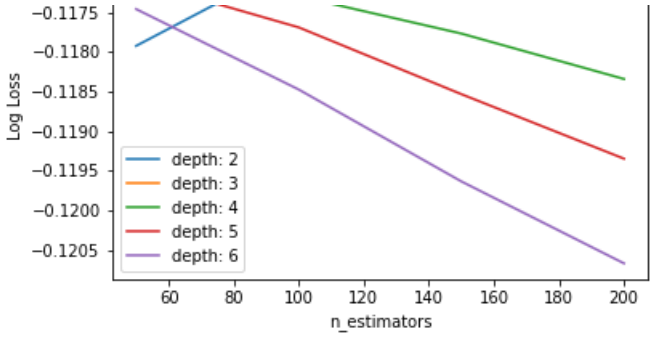
```
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot
import numpy
# load data
data = pd.read_csv('transactions_company_103600030.csv')
dataset = data.values
# split data into X and y
X = tx_class.drop(['LTVCluster','m6_Revenue'],axis=1)
y = tx_class['LTVCluster']
# encode string class values as integers
label_encoded_y = LabelEncoder().fit_transform(y)
# grid search
model = XGBClassifier()
n_estimators = [50, 100, 150, 200]
max_depth = [2, 3, 4, 5, 6]

print(max_depth)
param_grid = dict(max_depth=max_depth, n_estimators=n_estimators )
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold, verbose=
1)
grid_result = grid_search.fit(X, label_encoded_y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
 print("%f (%f) with: %r" % (mean, stdev, param))
# plot results
scores = numpy.array(means).reshape(len(max_depth), len(n_estimators))
for i, value in enumerate(max_depth):
    pyplot.plot(n_estimators, scores[i], label='depth: ' + str(value))
    pyplot.legend()
    pyplot.xlabel('n_estimators')
    pyplot.ylabel('Log Loss')
    pyplot.savefig('n_estimators_vs_max_depth.png')
```

```
[2, 3, 4, 5, 6]
Fitting 10 folds for each of 20 candidates, totalling 200 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 24 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   13.4s
[Parallel(n_jobs=-1)]: Done 152 tasks      | elapsed:  6.0min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:  8.7min finished
```

```
Best: -0.116666 using {'max_depth': 2, 'n_estimators': 150}
-0.117925 (0.001688) with: {'max_depth': 2, 'n_estimators': 50}
-0.116846 (0.001573) with: {'max_depth': 2, 'n_estimators': 100}
-0.116666 (0.001562) with: {'max_depth': 2, 'n_estimators': 150}
-0.116682 (0.001497) with: {'max_depth': 2, 'n_estimators': 200}
-0.117087 (0.001597) with: {'max_depth': 3, 'n_estimators': 50}
-0.116904 (0.001471) with: {'max_depth': 3, 'n_estimators': 100}
-0.117109 (0.001436) with: {'max_depth': 3, 'n_estimators': 150}
-0.117385 (0.001342) with: {'max_depth': 3, 'n_estimators': 200}
-0.117074 (0.001708) with: {'max_depth': 4, 'n_estimators': 50}
-0.117299 (0.001579) with: {'max_depth': 4, 'n_estimators': 100}
-0.117769 (0.001580) with: {'max_depth': 4, 'n_estimators': 150}
-0.118342 (0.001584) with: {'max_depth': 4, 'n_estimators': 200}
-0.117101 (0.001522) with: {'max_depth': 5, 'n_estimators': 50}
-0.117691 (0.001541) with: {'max_depth': 5, 'n_estimators': 100}
-0.118533 (0.001550) with: {'max_depth': 5, 'n_estimators': 150}
-0.119346 (0.001567) with: {'max_depth': 5, 'n_estimators': 200}
-0.117461 (0.001472) with: {'max_depth': 6, 'n_estimators': 50}
-0.118475 (0.001520) with: {'max_depth': 6, 'n_estimators': 100}
-0.119633 (0.001578) with: {'max_depth': 6, 'n_estimators': 150}
-0.120666 (0.001548) with: {'max_depth': 6, 'n_estimators': 200}
```





In [217]:

```
from sklearn.multiclass import OneVsRestClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import MultiLabelBinarizer

clf = OneVsRestClassifier(XGBClassifier(n_jobs=-1, max_depth=3, learning_rate=0.1, n_estimators=200
))
clf.fit(X_train, y_train)
```

Out[217]:

```
OneVsRestClassifier(estimator=XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=None, gamma=None,
                                            gpu_id=None, importance_type='gain',
                                            interaction_constraints=None,
                                            learning_rate=0.1,
                                            max_delta_step=None, max_depth=3,
                                            min_child_weight=None, missing=nan,
                                            monotone_constraints=None,
                                            n_estimators=200, n_jobs=-1,
                                            num_parallel_tree=None,
                                            random_state=None, reg_alpha=None,
                                            reg_lambda=None,
                                            scale_pos_weight=None,
                                            subsample=None, tree_method=None,
                                            validate_parameters=None,
                                            verbosity=None))
```

In [218]:

```
print('Accuracy of XGB classifier on training set: {:.2f}'
       .format(clf.score(X_train, y_train)))
print('*'*60)

print('Accuracy of XGB classifier on test set: {:.2f}'
       .format(clf.score(X_test[X_train.columns], y_test)))
print('*'*60)
pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Accuracy of XGB classifier on training set: 0.82
************************************************************
Accuracy of XGB classifier on test set: 0.81
************************************************************
              precision    recall  f1-score   support

           0       0.88      1.00      0.93     26332
           1       0.82      0.78      0.80     15556
           2       0.76      0.70      0.73      9708
           3       0.73      0.65      0.69      6314
           4       0.69      0.62      0.65      4200
           5       0.66      0.60      0.63      2754
           6       0.67      0.57      0.61      1807
           7       0.66      0.59      0.62      1207
           8       0.73      0.58      0.65       870
           9       0.86      0.70      0.77       509

    accuracy                           0.81     69257
```

```
   macro avg       0.75      0.68      0.71     69257
weighted avg       0.80      0.81      0.80     69257
```

In [349]:

```
segmented_data.head()
```

Out[349]:

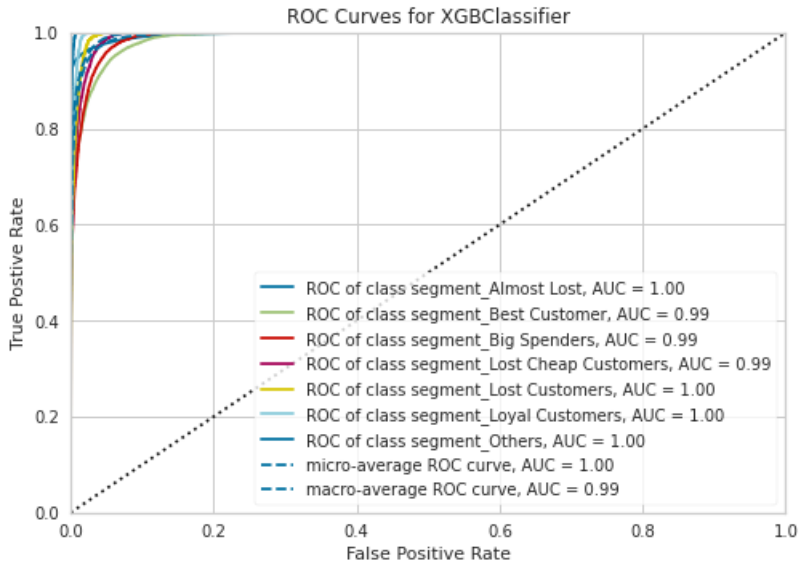
	id	recency	frequency	monetary_value	R	F	M	Cluster	monetary_value5M	segment_Almost Lost	segment_Best Customer	segment_Big Spenders
0	86246	-53	48	348.65	1	1	1	1	165.80	0	1	0
1	16412251	-65	35	137.64	1	1	1	1	110.92	0	1	0
2	16606739	54	10	111.00	3	1	1	2	111.00	1	0	0
3	17678929	-80	15	195.89	1	1	1	1	172.72	0	1	0
4	21024070	71	11	145.36	3	1	1	2	145.36	1	0	0

In [350]:

```
from yellowbrick.classifier import ROCAUC
visualizer = ROCAUC(ltv_xgb_model, classes=["segment_Almost Lost", "segment_Best Customer","segment
_Big Spenders", 'segment_Lost Cheap Customers', 'segment_Lost Customers', 'segment_Loyal
Customers', 'segment_Others'])

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.poof()
visualizer
```

Out[350]:

```
ROCAUC(ax=<AxesSubplot:title={'center':'ROC Curves for XGBClassifier'}, xlabel='False Positive Rat
e', ylabel='True Postive Rate'>,
       classes=['segment_Almost Lost', 'segment_Best Customer',
                'segment_Big Spenders', 'segment_Lost Cheap Customers',
                'segment_Lost Customers', 'segment_Loyal Customers',
                'segment_Others'],
       model=None)
```



In [355]:

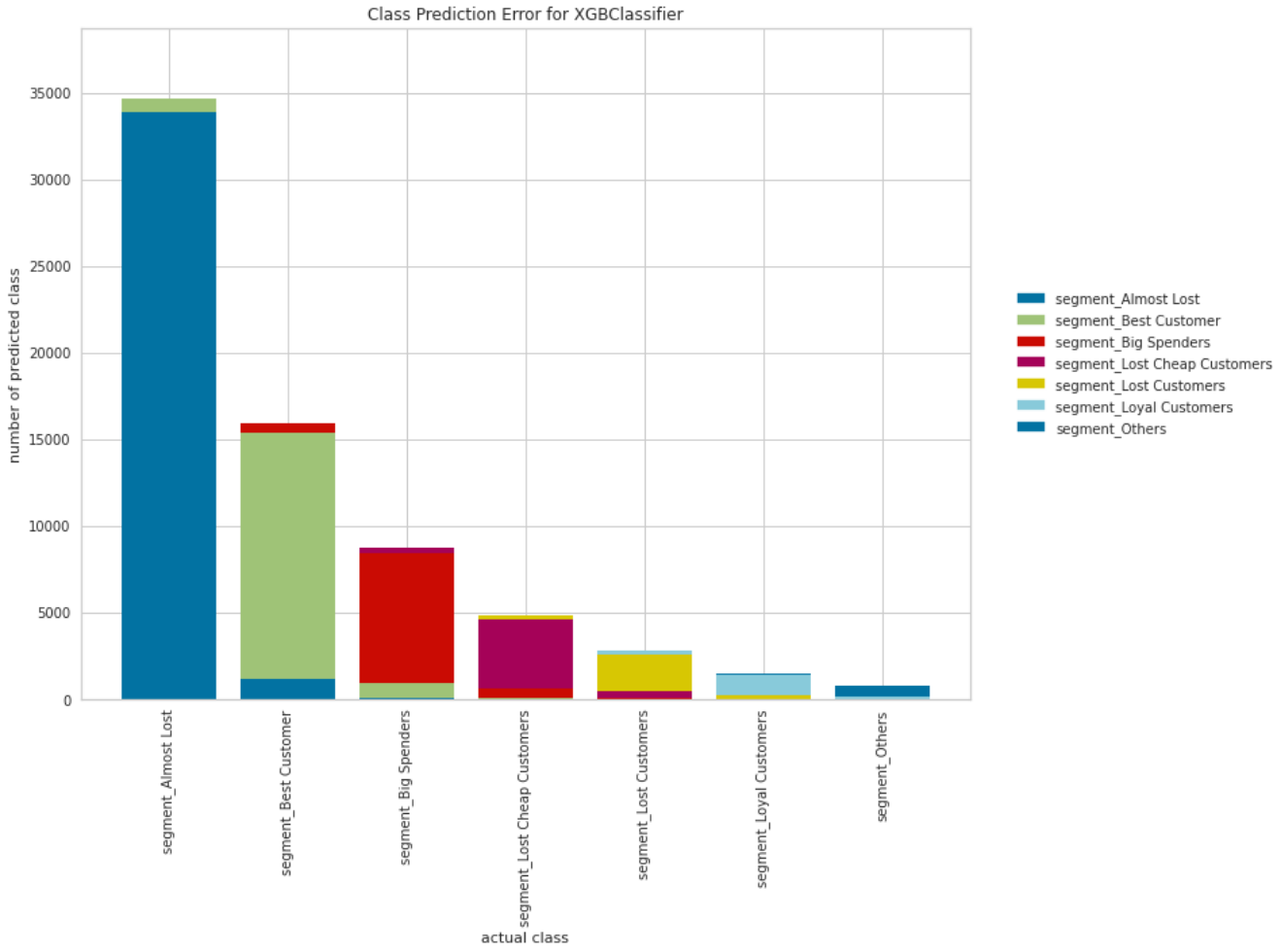
```
from yellowbrick.classifier import ClassPredictionError
visulizer = ClassPredictionError(XGBClassifier(random_sate = 42), classes=["segment_Almost Lost",
                                                                           "segment_Best Customer",
                                                                           "segment_Big Spenders",
```

```
                                                                'segment_Lost Cheap Cust
rs',
                                                                'segment_Lost Customers'
                                                                'segment_Loyal Customers
                                                                'segment_Others'], size=
80, 720))
visulizer.fit(X_train, y_train)
visulizer.score(X_test, y_test)
visulizer.poof()
```

```
[22:00:50] WARNING: ../src/learner.cc:516:
Parameters: { random_sate } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

Out[355]:

```
<AxesSubplot:title={'center':'Class Prediction Error for XGBClassifier'}, xlabel='actual class', y
label='number of predicted class'>
```

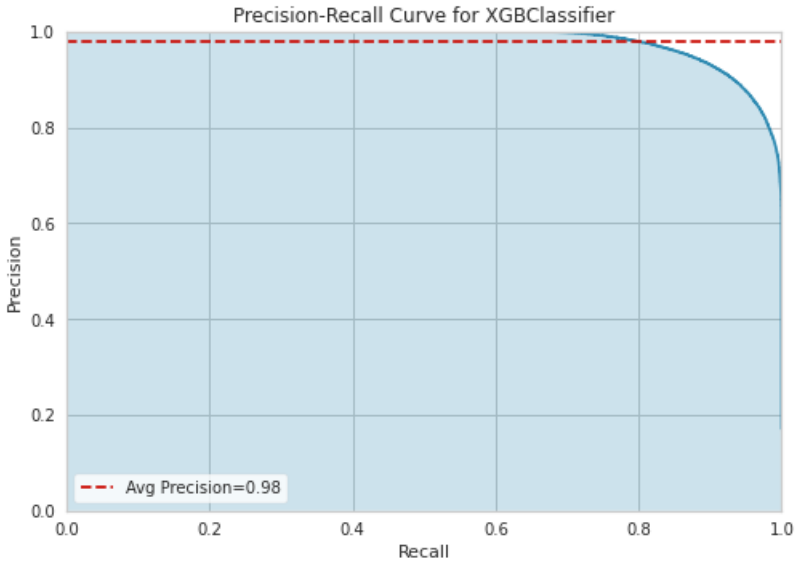


In [356]:

```
from yellowbrick.classifier import PrecisionRecallCurve
# Create the visualizer, fit, score, and poof it
viz = PrecisionRecallCurve(XGBClassifier())
viz.fit(X_train, y_train)
viz.score(X_test, y_test)
viz.poof()
```

Out[356]:

```
<AxesSubplot:title={'center':'Precision-Recall Curve for XGBClassifier'}, xlabel='Recall',
ylabel='Precision'>
```



In [99]:

```
from sklearn import metrics

# compute the R Square for model
print("R-Square:",metrics.r2_score(y_test, y_pred))

# calculate MAE using scikit-learn
print("MAE:",metrics.mean_absolute_error(y_test,y_pred))

#calculate mean squared error
print("MSE",metrics.mean_squared_error(y_test, y_pred))

# compute the RMSE of our predictions
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R-Square: 0.8437061234672792
MAE: 0.043374676927963964
MSE 0.044009991769785005
RMSE: 0.20978558522878785
```

In [353]:

```
from sklearn import metrics

# compute the R Square for model
print("R-Square:",metrics.r2_score(y_test, y_pred))

# calculate MAE using scikit-learn
print("MAE:",metrics.mean_absolute_error(y_test,y_pred))

#calculate mean squared error
print("MSE",metrics.mean_squared_error(y_test, y_pred))

# compute the RMSE of our predictions
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R-Square: 0.9400459402876156
MAE: 0.09124250956609631
MSE 0.11004259620244025
RMSE: 0.33172668901136104
```

In [ ]:

```
pareto_cutoff = tx_data['purchaseamount'].sum() * 0.8
print("The 80% of total revenue is: ",round(pareto_cutoff,2))
```

In [ ]:

```
pareto_cutoff = tx_clas['LTVCluster'] = 2
```

In [117]:

```
Test = pd.DataFrame(y_test)

Test.to_csv('Y_Test.csv')

Test.shape
```

Out[117]:

```
(69257, 1)
```

In [118]:

```
 Prediction = pd.DataFrame(y_pred)

Prediction.to_csv('Y_Prediction.csv')
```

In [ ]:

```
print("Best Customers: ",len(rfm_segmentation[rfm_segmentation['RFMScore']=='444']))
print('Loyal Customers: ',len(rfm_segmentation[rfm_segmentation['F_Quartile']==4]))
print("Big Spenders: ",len(rfm_segmentation[rfm_segmentation['M_Quartile']==4]))
print('Almost Lost: ', len(rfm_segmentation[rfm_segmentation['RFMScore']=='244']))
print('Lost Customers: ',len(rfm_segmentation[rfm_segmentation['RFMScore']=='144']))
print('Lost Cheap Customers: ',len(rfm_segmentation[rfm_segmentation['RFMScore']=='111']))
```